

交换机转发实验报告

实验内容

- 实现对数据结构mac_port_map的所有操作，以及数据包的转发和广播操作
 - iface_info_t *lookup_port(u8 mac[ETH_ALEN]);
 - void insert_mac_port(u8 mac[ETH_ALEN],iface_info_t *iface);
 - int sweep_aged_mac_port_entry();
 - void broadcast_packet(iface_info_t *iface,const char *packet,int len);
 - void handle_packet(iface_info_t *iface,char *packet,int len);
- 使用iperf和给定的拓扑进行实验，对比交换机转发与集线器广播的性能

实验流程

首先是对switch文件夹中所涉及的list.h,base.h等数据结构的熟悉，接下来分别介绍五个部分的写法：

lookup_port

```
iface_info_t *lookup_port(u8 mac[ETH_ALEN])
{
    // TODO:implement the lookup process here
    fprintf(stdout,"TODO:implement the lookup process here.\n");
    pthread_mutex_lock(&mac_port_map.lock);
    /*在正式的代码开始前上锁*/
    int position = 0;
    position = hash8(mac,ETH_ALEN);
    /*根据mac地址，确定从哈希表的哪一个下标所指示的双向链表中开始查找*/
    struct mac_port_entry *port = NULL;
    /*这里根据list_for_each_entry(pos,head,member)来定义了一个pos变量，pos需要设置为
    struct mac_port_entry*,
    就是表明指向双向链表内容结构体的指针，一个常见的错误是，由于后续pos 会有赋值，所以在初始时
    直接写成
    list_for_each_entry(NULL,head,member),但这与之前先声明一个结构体类型指针，再赋初值为
    NULL不同
    传入参数为 NULL 之后，对零地址进行赋值操作，直接产生段错误
    而声明一个 struct mac_port_entry *port 之后，为局部变量 port 分配了一个地址，只不过现在
    里面的内容是 NULL，但之后可以重新赋值*/
    list_for_each_entry(port,&mac_port_map.hash_table[position],list){
        /*这些写可以省略对 hash表 position 位置 为空的判断，因为这里是双向链表，而 head 又与其它的
        mac_port_entry 项不同，所以相对比较麻烦，如果用list.h里封装的,则 head -> next !=
        head 可以直接进行判断*/
        if(port -> mac[0] == mac[0] && port -> mac[1] == mac[1] && port->mac[2]
        == mac[2] && port->mac[3] == mac[3] && port->mac[4]==mac[4] && port-
        >mac[5]==mac[5]){
            /* 不能写成 port -> mac == mac ,mac的地址是外部传入的,port->mac的地址是hash表的，
            我们只能挨个比较内容*/
            pthread_mutex_unlock(&mac_port_map.lock);
            /*返回之前上锁*/
            return port->iface;
        }
    }
    /*返回之前上锁*/
    pthread_mutex_unlock(&mac_port_map.lock);
}
```

```

    /*打印 hash 表, 用于 debug*/
    dump_mac_port_table();
    return NULL;
}

```

具体的思路如下:

1. 首先查询操作与哈希表的查询一致, 只是需要注意数据结构上的细节。根据mac地址, mac地址是哈希表用来指示位置的指标, 调用hash8()函数求出其在哈希数组中的下标
2. 如果 mac_port_map 表头的 next 指针为NULL, 表明没有表项, 返回NULL
3. 但封装了list_for_each_entry()函数, 如果哈希数组的单链表为空, 则对于双向链表而言, 其第一个结点的next指针指向自己, for循环自动结束, 不需要再另行判断。
4. 表项非空, 对照mac地址是否相同。因为不同的mac地址可能映射到哈希表的同一个 position 的指标下, 所以需要对mac_port_map.hash_Table[position] 来进行轮询
5. <list.h>里封装的双向链表有头指针, 头指针不存储实际的数据, 只有前驱和后继两个指针, 这里hash表的表头就用来表示每一组双向链表的头指针
6. 锁操作, 在代码的临界区上锁和释放锁, 但需要注意的是, 由于 iface_info_t*函数返回一个指向iface_info_t 类型的结构体的指针, 所以在 return 之前必须要释放锁, 否则其它函数永远也拿不到这个被上了的锁。

insert_mac_port

```

void insert_mac_port(u8 mac[ETH_ALEN],iface_info_t *iface)
{
    // TODO:implement the insertion process here
    fprintf(stdout,"TODO:implement the insertion process here.\n");
    /*查询是否有目的mac 的端口信息*/
    iface_info_t *look_port = lookup_port(mac);
    /*查询之后上锁, 如果在之前上锁, 写出
    pthread_mutex_lock(&mac_port_map.lock);
    iface_info_t *look_port = lookup_port(mac)
    则在插入中上锁, 还没有解锁, 执行lookup时就拿不到锁, 所以无法指针, 卡住*/
    pthread_mutex_lock(&mac_port_map.lock);

    int position = 0;
    position = hash8(mac,ETH_ALEN);

    if(look_port == NULL){
        struct mac_port_entry *new = NULL;
        new = (struct mac_port_entry *)malloc(sizeof(struct mac_port_entry));
        int i = 0;
        for(i = 0; i < ETH_ALEN;i++){
            new ->mac[i] = mac[i];
        }
        new -> iface = iface;
        new -> visited = time(NULL);
        /*返回结果为空, 表明没查到, 声明新的结构体, 尾插法, 其头节点就是哈希表position位置对应的哈希表的表头, mac_port_map.hash_table[position],由于head是指针, 而结构体中 struct list_head list 是一个结构体, 所以需要取地址*/
        list_add_tail(new,&mac_port_map.hash_table[position]);
    }
    else{
        struct mac_port_entry *port = NULL;
        /*look_port 是 iface_info_t 的地址, 而不是 iface_info_t*的地址, 所以提供不了求偏移地址的有效信息, 需要复制lookup_port的代码端*/
        list_for_each_entry(port,&mac_port_map.hash_table[position],list){

```

```

        if(port -> mac[0] == mac[0] && port -> mac[1] == mac[1] && port-
>mac[2] == mac[2] && port->mac[3] == mac[3] && port->mac[4]==mac[4] && port-
>mac[5]==mac[5]){
            /*数组判断相等,更新, time(NULL)表明现在距离1900年的时间*/
            port -> iface = iface;
            port -> visited = time(NULL);
        }
    }
}
/*释放锁*/
pthread_mutex_unlock(&mac_port_map.lock);
}

```

具体的设计思路如下：

1. 添加表项的操作，传入的是端口号和mac地址，但第一步需要调用lookup_port函数来返回查询结果
2. 如果没找到，申请一块新的空间，然后插入，如果找到了，更新iface即端口信息和老化时间信息
3. 一开始的想法很美好，直接定义一个 `iface_info_t *look_port = lookup_port(mac);` 问题出在，这个look_port是否可以由它找到成员的地址呢？
`list_entry(ptr, type, member)`，是由其中的一个成员地址-偏移量=结构体地址
 但是这里，我们假设A 地址处，存储了iface_info_t *look_port, A 地址里的内容是B地址
 所以实际上传入的是B地址，B地址处存储 iface_info_t ，实际上我们想用通过
`insert_mac_port(iface_info_t* iface)` 得到的只是iface_info_t 结构体的地址，而真正能求出
`mac_port_entry`的是地址A，换言之，如果传入 `iface_info_t **iface` ,才能得到想要的结果
4. 我们得到了B地址，但A地址-偏移量才是结构体的地址，所以调用 `lookup_port`得不到非零指针时的有效信息
5. 所以在调用 `lookup_port`的时候，如果传入零地址，进行尾插法操作，如果非零，则需要将原来查询操作部分的全部代码复制，再更新 端口和老化信息。

sweep_aged_mac_port_entry

```

int sweep_aged_mac_port_entry()
{
    // TODO: implement the sweeping process here
    printf("TODO:implement the sweeping process here.\n");
    pthread_mutex_lock(&mac_port_map.lock);
    /* u8 i = 0 会产生死循环 */
    int i = 0;
    for(i = 0; i < HASH_8BITS; i++){
        if(list_empty(mac_port_map.hash_table[i].next)){
            continue;
        }

        struct mac_port_entry *find_port;
        struct mac_port_entry *q;
        /* 定义 find_port 和 q 只起到分配地址的作用，如果
        list_for_each_entry_safe(NULL, NULL, head, list),则会导致给0地址赋值的段错误*/
        list_for_each_entry_safe(find_port, q, &mac_port_map.hash_table[i], list){
            if(time(NULL) - find_port -> visited >= MAC_PORT_TAMEOUT){
                /*调用 list.h 里的函数*/
                list_delete_entry(&find_port->list);
                free(find_port);
            }
        }
    }
}
pthread_mutex_unlock(&mac_port_map.lock);

```

```

    return 0;
}

```

具体的设计思路如下:

1. 对每个非空的双向链表进行遍历, 采用封装的 `list_for_each_entry_safe()` 进行删除操作
2. 特别注意, 一开始出现了一个bug, 使得for循环变为死循环, 因为设置 `i` 为 `u8` 类型, 哈希表长度恰为 `255`, `255 + 1 == 0`, 从而导致循环从不跳出, 改进的方法是将 `i` 设为 `int` 类型。

handle_packet

```

void handle_packet(iface_info_t *iface, char *packet, int len)
{
    // TODO: implement the packet forwarding process here
    fprintf(stdout, "TODO: implement the packet forwarding process here.\n");
    /*到 insert 之前属于更新表项, 根据packet解析源地址, 将源地址插入到哈希表中,
    由于在insert_mac_port中, 已经考虑到了传入的mac地址在哈希表中: 有对应项, 没有对应项:
    有对应项但不相等的情况, 所以这里不需要额外的判断*/
    struct ether_header *eh = (struct ether_header *)packet;
    log(DEBUG, "the dst mac address is " ETHER_STRING ".\n", ETHER_FMT(eh->ether_dhost));
    insert_mac_port(eh->ether_shost, iface);

    /*目的地址的查找操作: 如果找到就发送, 否则广播, 查询目的地址不需要更新老化等操作*/
    iface_info_t *lookup;
    lookup = lookup_port(eh-> ether_dhost);
    if(lookup != NULL){
        iface_send_packet(lookup, packet, len);
    }
    else{
        broadcast_packet(iface, packet, len);
    }
    free(packet);
}

```

broadcast_packet

这也就是广播网络实验的实验内容

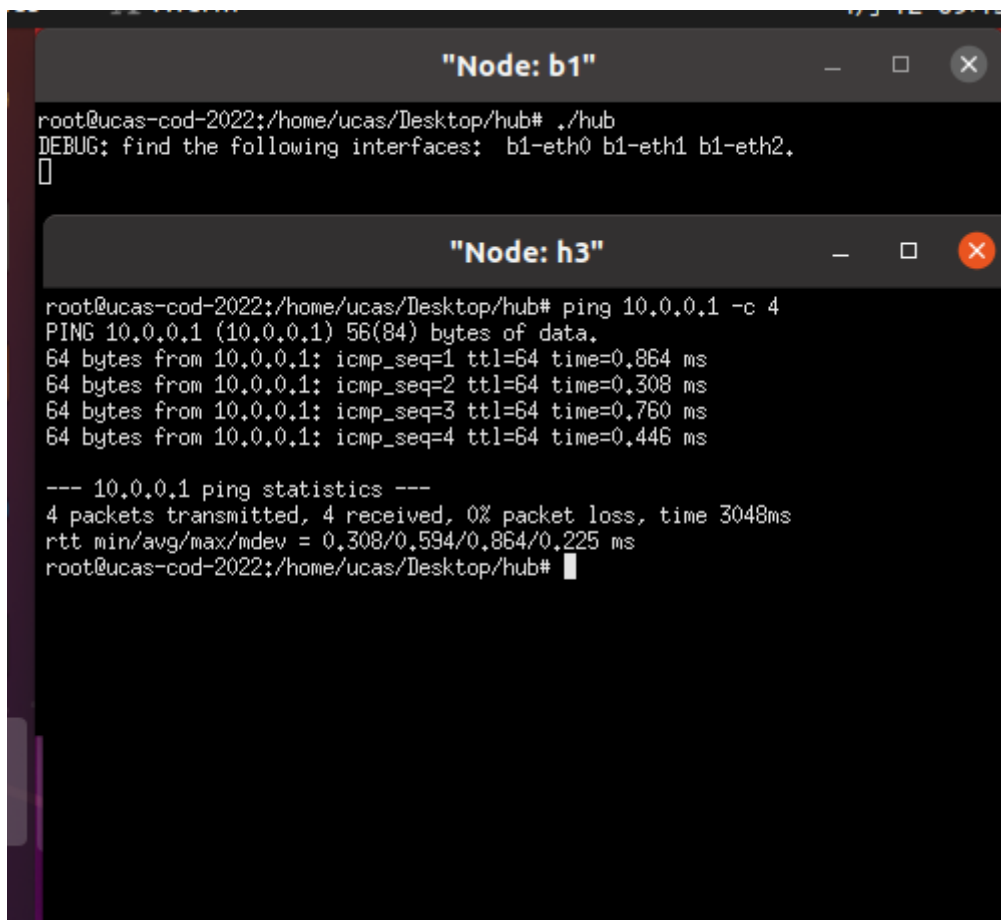
```

void broadcast_packet(iface_info_t *iface, const char *packet, int len)
{
    //TODO: broadcast packet
    fprintf(stdout, "TODO: broadcast packet.\n");
    iface_info_t *current = iface;
    /*这里对结构体进行说明, base.h里封装了两个结构体 ustack_t 和 iface_info_t,
    对于每一个主机或者交换机而言, 我只知道本机的端口信息, ustack_t 中的 iface_list
    成员存放了端口信息链表的头节点指针, iface_info_t 是每一个结点的信息*/
    list_for_each_entry(iface, &instance->iface_list, list){
        /*从头结点遍历, 从不是接收端的端口广播*/
        if(iface != current){
            iface_send_packet(iface, packet, len);
        }
    }
}

```

实验结果

广播网络实验



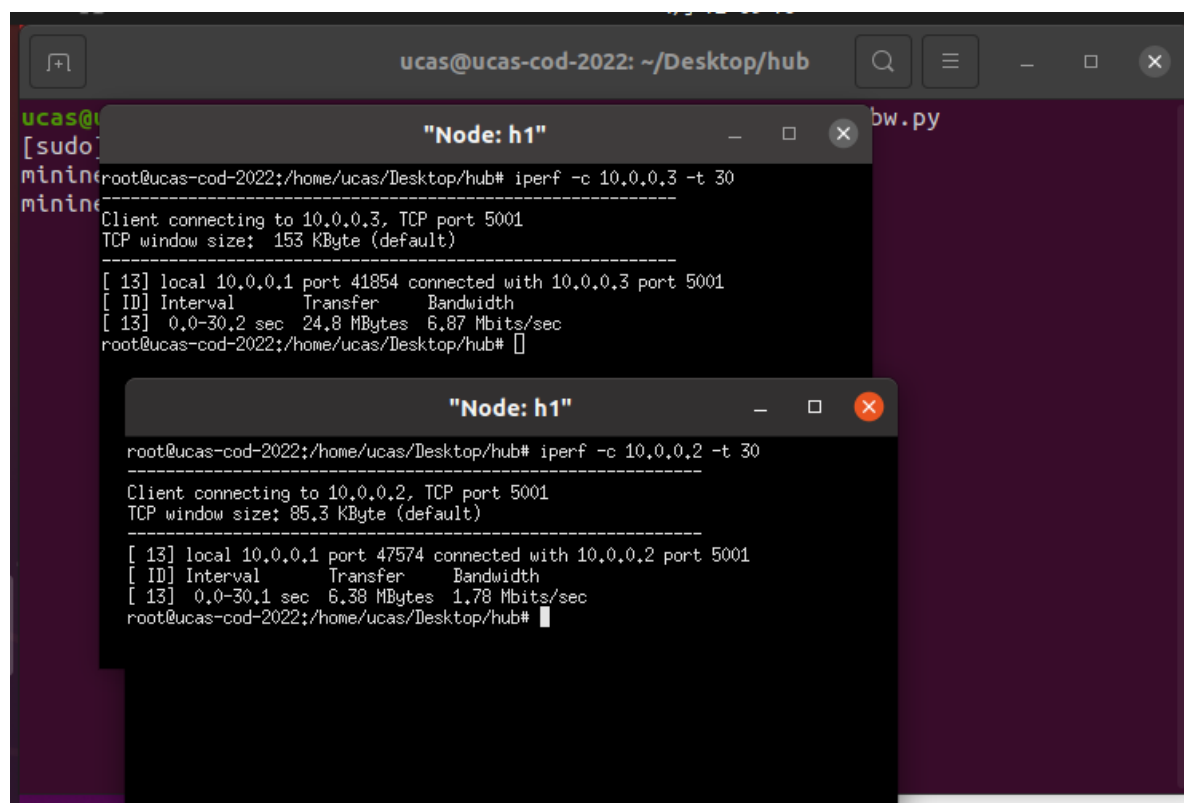
The image shows two terminal windows. The top window, titled "Node: b1", shows a command prompt where the user runs `./hub`. The output indicates that the program found interfaces `b1-eth0`, `b1-eth1`, and `b1-eth2`. The bottom window, titled "Node: h3", shows a command prompt where the user runs `ping 10.0.0.1 -c 4`. The output shows four successful ping requests to `10.0.0.1` with varying response times. Below the ping results, a summary statistics line is shown: `--- 10.0.0.1 ping statistics ---`, followed by `4 packets transmitted, 4 received, 0% packet loss, time 3048ms` and `rtt min/avg/max/mdev = 0.308/0.594/0.864/0.225 ms`.

```
root@ucas-cod-2022:/home/ucas/Desktop/hub# ./hub
DEBUG: find the following interfaces: b1-eth0 b1-eth1 b1-eth2.
[]

root@ucas-cod-2022:/home/ucas/Desktop/hub# ping 10.0.0.1 -c 4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.864 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.308 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.760 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.446 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3048ms
rtt min/avg/max/mdev = 0.308/0.594/0.864/0.225 ms
root@ucas-cod-2022:/home/ucas/Desktop/hub#
```

- 图1: 表明实现了 `broadcast_packet`函数, 实现从 `h3` 结点向 `h1` 端口成功发送数据。



The image shows two terminal windows. The top window, titled "Node: h1", shows a command prompt where the user runs `iperf -c 10.0.0.3 -t 30`. The output shows a client connecting to `10.0.0.3` on TCP port 5001, and a summary line: `[13] 0.0-30.2 sec 24.8 MBytes 6.87 Mbits/sec`. The bottom window, titled "Node: h3", shows a command prompt where the user runs `iperf -c 10.0.0.2 -t 30`. The output shows a client connecting to `10.0.0.2` on TCP port 5001, and a summary line: `[13] 0.0-30.1 sec 6.38 MBytes 1.78 Mbits/sec`.

```
root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.3 -t 30
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 153 KByte (default)
-----
[ 13] local 10.0.0.1 port 41854 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.2 sec  24.8 MBytes 6.87 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#

root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.2 -t 30
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.1 port 47574 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.1 sec  6.38 MBytes 1.78 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
```

The screenshot shows a terminal window titled 'ucas@ucas-cod-2022: ~/Desktop/hub'. The user runs 'sudo python2 three_nodes_bw.py'. The output shows 'mininet> x' and 'mininet> q'. Two terminal windows titled '"Node: h1"' are shown. The first window shows the command 'root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.2 -t 30' and the output: 'Client connecting to 10.0.0.2, TCP port 5001', 'TCP window size: 144 KByte (default)', and a table showing a bandwidth of 2.79 Mbits/sec. The second window shows the command 'root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.3 -t 30' and the output: 'Client connecting to 10.0.0.3, TCP port 5001', 'TCP window size: 85.3 KByte (default)', and a table showing a bandwidth of 5.41 Mbits/sec.

```
ucas@ucas-cod-2022:~/Desktop/hub$ sudo python2 three_nodes_bw.py
[sudo] password for ucas:
mininet> x
mininet> q
ucas@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.2 -t 30
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 144 KByte (default)
-----
[ 13] local 10.0.0.1 port 47576 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.8 sec  10.2 MBytes  2.79 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.3 -t 30
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.1 port 41856 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.8 sec  19.9 MBytes  5.41 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
```

- 图2：进行 iperf测试，这里h1 作为客户端，向 h2 和 h3 服务器端发送数据，由于理论上，b1 到 h2 和 h3 的网络分别为 10 Mb/s，但这里（虽然由于输入命令有一定的时间间隔），但h2 为 1.78Mb/s, h3 为 6.87Mb/s,都显著低于应有传输速率，利用率只有 40 %。
- 图3：和图2 相同的测试，h2 结点为 2.79 Mb/s, h3 结点为 5.41 Mb/s,利用率仍约 40%。

The screenshot shows a terminal window titled 'ucas@ucas-cod-2022: ~/Desktop/hub'. The user runs 'sudo python2 three_nodes_bw.py'. The output shows 'mininet> x' and 'mininet> q'. Two terminal windows titled '"Node: h2"' and '"Node: h3"' are shown. The first window shows the command 'root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.1 -t 30' and the output: 'Client connecting to 10.0.0.1, TCP port 5001', 'TCP window size: 85.3 KByte (default)', and a table showing a bandwidth of 7.50 Mbits/sec. The second window shows the command 'root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.1 -t 30' and the output: 'Client connecting to 10.0.0.1, TCP port 5001', 'TCP window size: 170 KByte (default)', and a table showing a bandwidth of 7.55 Mbits/sec.

```
ucas@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.2 port 45830 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.2 sec  27.0 MBytes  7.50 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 170 KByte (default)
-----
[ 13] local 10.0.0.3 port 36636 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.4 sec  27.4 MBytes  7.55 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
```

The screenshot shows three terminal windows. The top window, titled "Node: h1", shows the server listening on port 5001 and receiving two connections from 10.0.0.3 and 10.0.0.2, with bandwidths of 7.66 and 8.07 Mb/s respectively. The middle window, titled "Node: h2", shows a client connecting to 10.0.0.1 on port 5001 and receiving a connection from 10.0.0.2 with a bandwidth of 8.29 Mb/s. The bottom window, titled "Node: h3", shows a client connecting to 10.0.0.1 on port 5001 and receiving a connection from 10.0.0.3 with a bandwidth of 7.93 Mb/s.

```
ucas@ucas-cod-2022: ~/Desktop/hub
[Node: h1]
root@ucas-cod-2022:/home/ucas/Desktop/hub# ./hub
DEBUG: find the following interfaces: b1-eth0 b1-eth1 b1-eth2.
root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 14] local 10.0.0.1 port 5001 connected with 10.0.0.3 port 36638
[ 15] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 45832
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-31.6 sec  28.9 MBytes  7.66 Mbits/sec
[ 15] 0.0-31.3 sec  30.1 MBytes  8.07 Mbits/sec
[Node: h2]
root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 13] local 10.0.0.2 port 45832 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.5 sec  30.1 MBytes  8.29 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
[Node: h3]
root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 153 KByte (default)
[ 13] local 10.0.0.3 port 36638 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.5 sec  28.9 MBytes  7.93 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
```

The screenshot shows three terminal windows. The top window, titled "Node: h3", shows a client connecting to 10.0.0.1 on port 5001 and receiving a connection from 10.0.0.3 with a bandwidth of 7.93 Mb/s. The middle window, titled "Node: h2", shows a client connecting to 10.0.0.1 on port 5001 and receiving a connection from 10.0.0.2 with a bandwidth of 8.45 Mb/s. The bottom window, titled "Node: h1", shows the server listening on port 5001 and receiving two connections from 10.0.0.3 and 10.0.0.2, with bandwidths of 7.73 and 8.28 Mb/s respectively. A file named "3.31_data.sock" is visible in the background.

```
ucas@ucas-cod-2022: ~/Desktop/hub
[Node: h3]
root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 119 KByte (default)
[ 13] local 10.0.0.3 port 36640 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.4 sec  28.8 MBytes  7.93 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
[Node: h2]
root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 13] local 10.0.0.2 port 45834 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.0 sec  30.2 MBytes  8.45 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
[Node: h1]
root@ucas-cod-2022:/home/ucas/Desktop/hub# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 14] local 10.0.0.1 port 5001 connected with 10.0.0.3 port 36640
[ 15] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 45834
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-31.2 sec  28.8 MBytes  7.73 Mbits/sec
[ 15] 0.0-30.6 sec  30.2 MBytes  8.28 Mbits/sec
root@ucas-cod-2022:/home/ucas/Desktop/hub#
```

- 图4：h1 作为服务器，h2 和 h3 请求，h1 链路理论传输速率为 20Mb/s,现在 h2 和 h3 分别为 7.50 Mb/s 和 7.55 Mb/s，利用率为 75.25 %。
- 图5：分别得到 h2 处 8.29 Mb/s 和 h3 处 7.93 Mb/s,利用率为 81.1 %。图6：利用率:81.9

cycle_nodes_bw

环状拓扑网络测试 (分别有 h1-b1/ b3\ b2 - h2 这 5 个结点)

Capturing from h1-eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
2788...	14.073473014	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.073507479	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
2788...	14.073541934	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.073576815	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
2788...	14.073611000	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.073645261	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
2788...	14.073679685	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.073714595	3a:46:8f:9f:0b:10	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at 3a:46:8f:9f:0b:10
2788...	14.073746373	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.073780597	3a:46:8f:9f:0b:10	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at 3a:46:8f:9f:0b:10
2788...	14.073812118	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.074075794	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
2788...	14.074112424	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.074147676	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
2788...	14.074181956	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.074216876	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.074249433	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
2788...	14.074283744	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
2788...	14.074317404	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
2788...	14.074351697	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface h1-eth0, id 0
Ethernet II, Src: ce:f4:5b:6e:fe:2a (ce:f4:5b:6e:fe:2a), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)

```
root@ucas-cod-2022:/home/ucas/Desktop/hub# ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.157 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.157/0.157/0.157/0.000 ms
root@ucas-cod-2022:/home/ucas/Desktop/hub#
```

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
6426...	35.472205231	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472219490	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472225381	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472239778	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472245825	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472416772	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472424463	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472441159	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472448223	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472462824	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472468995	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472483528	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472489253	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472660843	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472668376	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472684915	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472691017	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472705884	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472711741	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
6426...	35.472726867	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface h1-eth0, id 0
Ethernet II, Src: ce:f4:5b:6e:fe:2a (ce:f4:5b:6e:fe:2a), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)

No.	Time	Source	Destination	Protocol	Length	Info
9410...	50.591314231	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
9410...	50.591320174	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
9410...	50.591348527	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
9410...	50.591373641	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
9410...	50.591596868	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
9410...	50.591607938	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
9410...	50.591614697	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
9410...	50.591620921	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
9410...	50.591649959	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
9411...	50.591672590	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
9411...	50.591695899	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
9411...	50.591703930	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
9411...	50.591710033	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
9411...	50.591716557	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50
9411...	50.591772728	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
9411...	50.591799235	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
9411...	50.591820939	ce:f4:5b:6e:fe:2a	3a:46:8f:9f:0b:10	ARP	42	10.0.0.1 is at ce:f4:5b:6e:fe:2a
9411...	50.591843253	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
9411...	50.592007908	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x167b, seq=1/256,
9411...	50.592016933	c2:d9:ce:38:3f:50	ce:f4:5b:6e:fe:2a	ARP	42	10.0.0.2 is at c2:d9:ce:38:3f:50

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface h1-eth0, id 0
 Ethernet II, Src: ce:f4:5b:6e:fe:2a (ce:f4:5b:6e:fe:2a), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 Address Resolution Protocol (request)

- 图7、8、9 表明：构成环状网络拓扑之后，发包成功，但是会一直循环，产生广播风暴。wireshark抓包的表项不断增长。

交换机实验

交换机实验没有解决广播风暴问题，这里利用环状拓扑的测试文件 cycle_nodes_bw.py 并定义了一个全局变量 count,来考察 switch 中调用 insert_mac_port 的次数：

```

Activities  Xterm  4月12 14:22

"Node: b2"
8e:31:60:db:3b:0f -> b2-eth0, 0
7a:00:80:4c:40:56 -> b2-eth1, 0
count=====15391
TODO: implement the lookup process here.
TODO: implement the packet forwarding process here.
DEBUG: the dst mac address is 7a:00:80:4c:40:56.

TODO: implement the insertion process here.
TODO: implement the lookup process here.
dumping the mac_port table:
8e:31:60:db:3b:0f -> b2-eth0, 0
7a:00:80:4c:40:56 -> b2-eth1, 0
count=====15392
TODO: implement the lookup process here.
TODO: implement the packet forwarding process here.
DEBUG: the dst mac address is 7a:00:80:4c:40:56.

TODO: implement the insertion process here.
TODO: implement the lookup process here.
dumping the mac_port table:
8e:31:60:db:3b:0f -> b2-eth0, 0
7a:00:80:4c:40:56 -> b2-eth1, 0
count=====15393
TODO: implement the lookup process here.

"Node: b3"
TODO: implement the packet forwarding process here.
DEBUG: the dst mac address is 7a:00:80:4c:40:56.

TODO: implement the insertion process here.
TODO: implement the lookup process here.
dumping the mac_port table:
8e:31:60:db:3b:0f -> b3-eth1, 0
7a:00:80:4c:40:56 -> b3-eth0, 0
count=====15187
TODO: implement the lookup process here.
TODO: implement the packet forwarding process here.
DEBUG: the dst mac address is 7a:00:80:4c:40:56.

TODO: implement the insertion process here.
TODO: implement the lookup process here.
dumping the mac_port table:
8e:31:60:db:3b:0f -> b3-eth1, 0
7a:00:80:4c:40:56 -> b3-eth0, 0
count=====15188
TODO: implement the lookup process here.
TODO: implement the packet forwarding process here.
DEBUG: the dst mac address is 7a:00:80:4c:40:56.

TODO: implement the insertion process here.

"Node: h1"
root@ucas-cod-20221/home/ucas/Desktop/switch# ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.24 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/ndev = 1.237/1.237/1.237/0.000 ms
root@ucas-cod-20221/home/ucas/Desktop/switch#

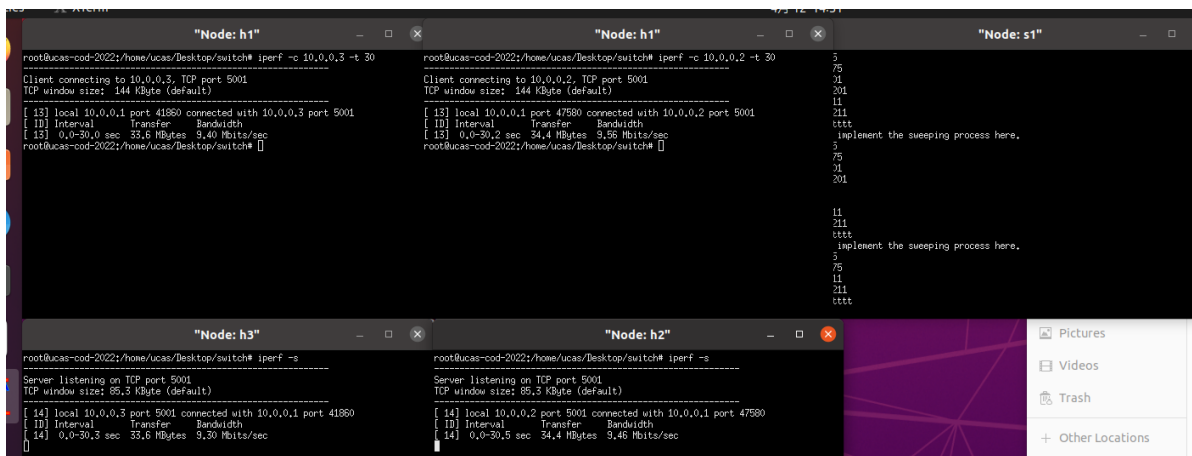
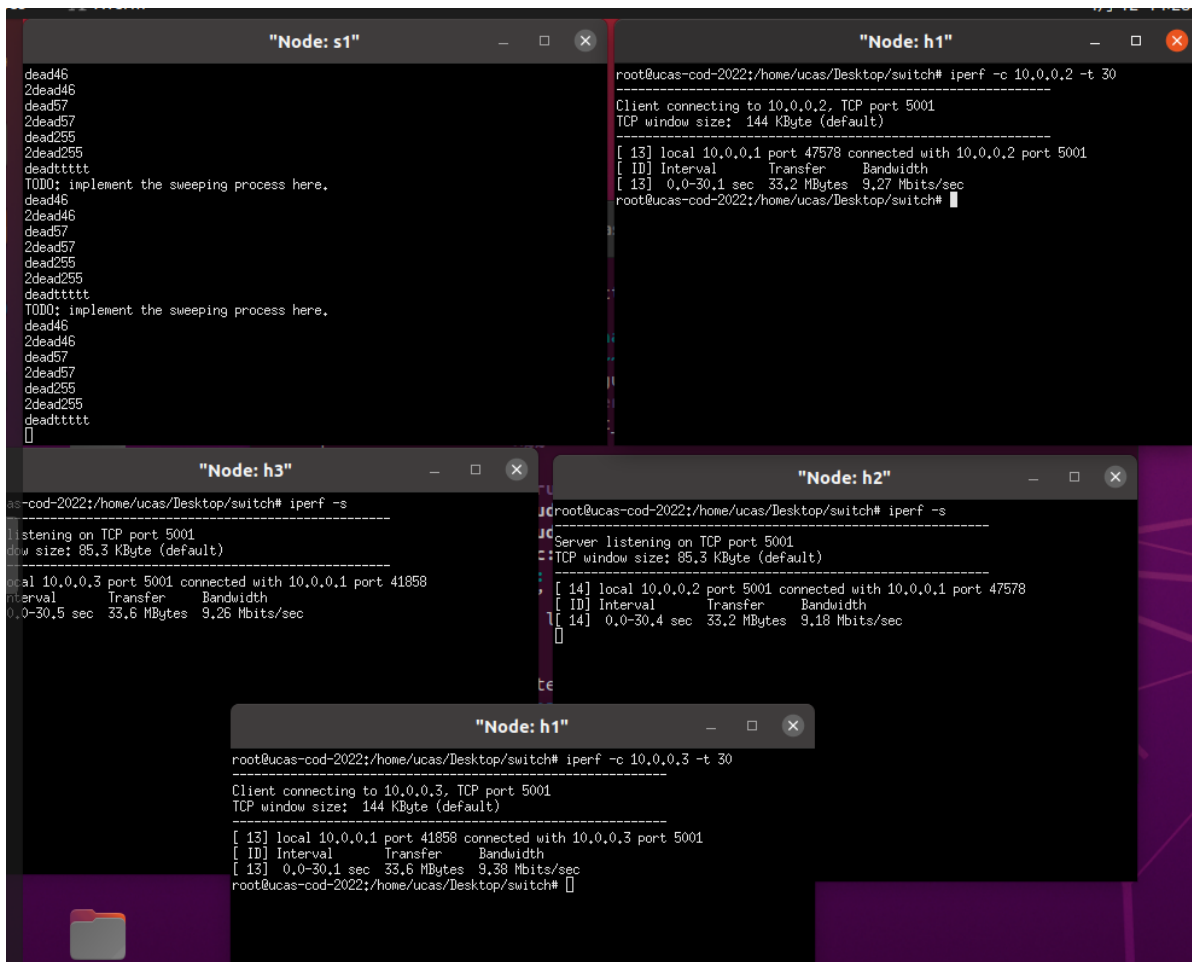
"Node: b1"
TODO: implement the insertion process here.
TODO: implement the lookup process here.
dumping the mac_port table:
8e:31:60:db:3b:0f -> b1-eth1, 0
7a:00:80:4c:40:56 -> b1-eth2, 0
count=====16106
TODO: implement the lookup process here.
TODO: implement the packet forwarding process here.
DEBUG: the dst mac address is 7a:00:80:4c:40:56.

TODO: implement the insertion process here.
TODO: implement the lookup process here.
dumping the mac_port table:
8e:31:60:db:3b:0f -> b1-eth1, 0
7a:00:80:4c:40:56 -> b1-eth2, 0
count=====16107
TODO: implement the lookup process here.
TODO: implement the packet forwarding process here.
DEBUG: the dst mac address is 7a:00:80:4c:40:56.

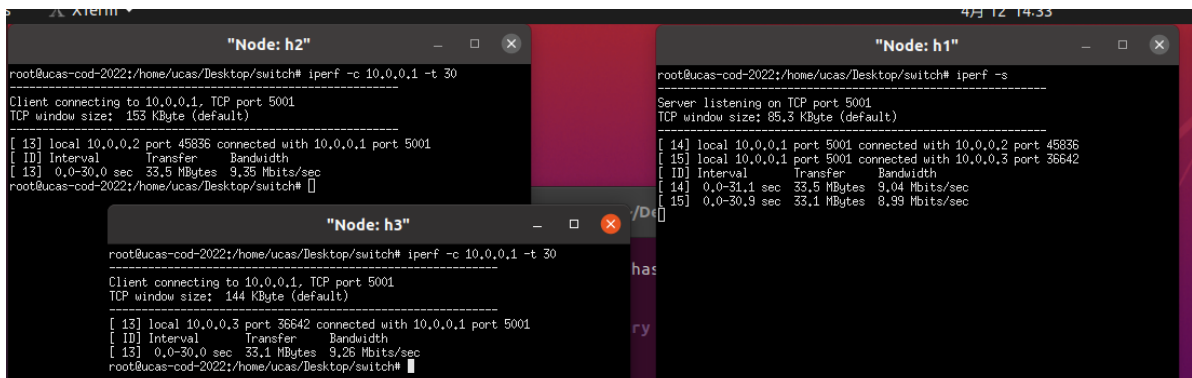
TODO: implement the insertion process here.

```

- 图10：可以看到发包成功，但count 计数已经达到 15000+ 次的查询次数，这个学习表并没有完全完善功能。



- 图11：在three_nodes_bw的测试拓扑下，h1 为客户端，分别向 h2 和 h3 发送消息，传输结果为 9.27 Mb/s 和 9.38 Mb/s，利用率显著提高，降低了不必要的传输时间。
- 图12：传输速率为 9.30Mb/s 和 9.46 Mb/s，利用率显著提高。



- 图 13: h1 为服务器端，h2 和 h3 为客户端，传输速率为 9.35Mb/s 和 9.26Mb/s。

思考题

1. 交换机在转发数据包时有两个查表操作：根据源MAC地址、根据目的MAC地址，为什么在查询源MAC地址时更新老化时间，查询目的MAC地址时不更新？

解：改变源MAC地址可以很快反映出拓扑结构的变化。如果根据目的MAC地址更新老化时间，假设拓扑结构变动，则发送的数据包无法到达目的MAC地址。这里可以考虑在查询操作之后，目的MAC地址改变。则由于更新了老化时间，查询原来的错误目的MAC地址这一表项一直被更新，导致一直留在hash表中。但由于接收端已经换了位置，所以收不到包，得不到接收确认的信号，导致死循环。如果不更新，则错误地址所在表项可以在30s之后被删除，恢复正常工作。

2. 网络中存在广播包，即发往网内所有主机的数据包，其目的MAC地址设置为全0xFF，例如ARP请求数据包。这种广播包对交换机转发表逻辑有什么影响？

解：帮助交换机完成转发表的构建和学习过程，因为不知道对方的MAC地址，所以广播。当对方收到后，会发送ARP包返回，此时根据发送信息完善转发表，最终完成转发表的动态构建。

3. 理论上，足够多个交换机可以连接起全世界所有的终端。请问，使用这种方式连接亿万台主机是否技术可行？并说明理由。

解：不可行。交换机过多，导致维护的表项数据量过大，查询操作、删除操作、插入操作等都需要巨大的维护成本，花费巨大的时间代价。